# Assignment 2 Report

Mannem S V Sayi Teja Reddy

ME20B108

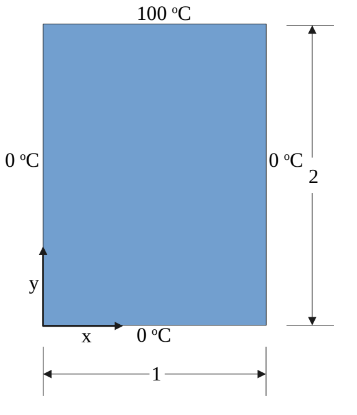
## PROBLEM STATEMENT

Consider a rectangular plate of unit thickness with dimensions 1 x 2 units (2-D steady state diffusion problem, Boundary value problem). Plot the steady state temperature contour of the rectangular plate for the following boundary conditions.

## GOVERNING EQUATION

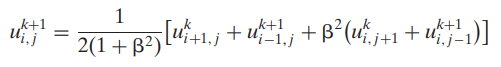
## BOUNDARY CONDITIONS

Top wall =

All other walls = 

## NUMERICAL FORMULATION



**Point Gauss-Seidel Iteration Method:** 

**Line Gauss-Seidel Iteration Method** 

**Point Successive Over-Relaxation Method (PSOR)**



1<ω<2 for over relaxation

0<ω<1 for under relaxation

**Line Successive Over-Relaxation Method (LSOR)**



1<ω<2 for over relaxation

0<ω<1 for under relaxation

**Alternating Direction Implicit (ADI) Method**





**Alternating Direction Implicit (ADI) Method with relaxation**



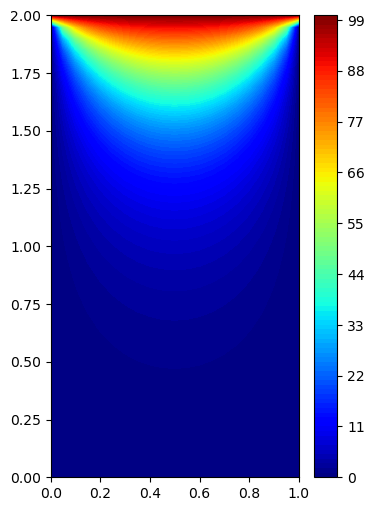


1<ω<2 for over relaxation

0<ω<1 for under relaxation

## RESULTS AND DISCUSSION

The steady state temperature contour of the rectangular plate



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.No. | Scheme used | Grid size | No. of Iteration | CPU time | Total Computational Time |
| 1 | Point Gauss Seidel | 21 x 41 | 593 | 1.91 s | 2.29 s |
| 2 | Line Gauss Seidel | 21 x 41 | 300 | 1.42 s | 1.67 s |
| 3 | PSOR | 21 x 41 | 72 | 359 ms | 572 ms |
| 4 | LSOR | 21 x 41 | 53 | 453 ms | 573 ms |
| 5 | ADI | 21 x 41 | 144 | 1.12 s | 1.41 s |
| 6 | ADI with relaxation | 21 x 41 | 75 | 641 ms | 971 ms |

## APPENDIX: Computer code

### Point Gauss-Seidel

import numpy as np

import matplotlib.pyplot as plt

row, col = (41, 21)

T = np.zeros((row,col),dtype=float)

T[40,:] = 100

tolerance = 0.01

dx = 1/(col-1)

dy = 2/(row-1)

beta = dx/dy

error = 1

iter = 0

while error > tolerance:

error = 0

iter += 1

for i in range(1,row-1):

for j in range(1,col-1):

T\_old = T[i,j]

T[i,j] = (T[i+1,j] + T[i-1,j] + beta\*\*2\*(T[i,j+1] + T[i,j-1]))/(2\*(1+beta\*\*2))

error += np.abs(T[i,j]-T\_old)

print('Number of iterations:',iter)

plt.figure(figsize=(4,6))

X,Y = np.meshgrid(np.linspace(0,1,col),np.linspace(0,2,row))

plt.contourf(X,Y,T,100,cmap='jet')

plt.colorbar()

plt.show()

### Line Gauss-Seidel

import numpy as np

import matplotlib.pyplot as plt

row, col = (41, 21)

T = np.zeros((row,col),dtype=float)

T[40,:] = 100

tolerance = 0.01

dx = 1/(col-1)

dy = 2/(row-1)

beta = dx/dy

error = 1

iter = 0

while error > tolerance:

error = 0

iter += 1

for i in range(1,row-1):

for j in range(1,col-1):

T\_old = T[i,j]

T[i,j] = (T[i+1,j] + T[i-1,j] + beta\*\*2\*(T[i,j+1] + T[i,j-1]))/(2\*(1+beta\*\*2))

error += np.abs(T[i,j]-T\_old)

print('Number of iterations:',iter)

plt.figure(figsize=(4,6))

X,Y = np.meshgrid(np.linspace(0,1,col),np.linspace(0,2,row))

plt.contourf(X,Y,T,100,cmap='jet')

plt.colorbar()

plt.show()

### PSOR

import numpy as np

import matplotlib.pyplot as plt

row, col = (41, 21)

T = np.zeros((row,col),dtype=float)

T[40,:] = 100

tolerance = 0.01

dx = 1/(col-1)

dy = 2/(row-1)

beta = dx/dy

omega = 1.781

error = 1

iter = 0

while error > tolerance:

error = 0

iter += 1

for i in range(1,row-1):

for j in range(1,col-1):

T\_old = T[i,j]

T[i,j] = (1 - omega) \* T[i,j] + omega \* ((T[i+1,j] + T[i-1,j] + beta\*\*2\*(T[i,j+1] + T[i,j-1]))/(2\*(1+beta\*\*2)))

error += np.abs(T[i,j]-T\_old)

print('Number of iterations:',iter)

plt.figure(figsize=(4,6))

X,Y = np.meshgrid(np.linspace(0,1,col),np.linspace(0,2,row))

plt.contourf(X,Y,T,100,cmap='jet')

plt.colorbar()

plt.show()

### LSOR

import numpy as np

import matplotlib.pyplot as plt

row, col = (41, 21)

T = np.zeros((row,col), dtype=np.float64)

T[40,:] = 100

tolerance = 0.01

omega = 1.275

dx = 1/(col-1)

dy = 2/(row-1)

beta = dx/dy

def tridiagonal\_solve(a, b, c, d):

n = len(d)

for i in range(1, n):

m = a[i-1] / b[i-1]

b[i] = b[i] - m \* c[i-1]

d[i] = d[i] - m \* d[i-1]

x = np.zeros(n)

x[-1] = d[-1] / b[-1]

for i in range(n-2, -1, -1):

x[i] = (d[i] - c[i] \* x[i+1]) / b[i]

return x

error = 1

iter = 0

a = np.zeros(col-3, dtype=np.float64)

b = np.zeros(col-2, dtype=np.float64)

c = np.zeros(col-3, dtype=np.float64)

d = np.zeros(col-2, dtype=np.float64)

while error > tolerance:

error = 0

iter += 1

T\_old = T.copy()

for i in range(1,row-1):

d[0] = -(beta\*\*2)\*omega\*(T[i-1,1] + T[i+1,1]) - omega\*T[i,0] - (1-omega)\*2\*(1+beta\*\*2)\*T[i,1]

d[-1] = -(beta\*\*2)\*omega\*(T[i-1,col-2] + T[i+1,col-2]) - omega\*T[i,col-1] - (1-omega)\*2\*(1+beta\*\*2)\*T[i,col-2]

b[0] = -2\*(1+beta\*\*2)

b[-1] = -2\*(1+beta\*\*2)

for j in range(1,col-3):

a[j-1] = omega

b[j] = -2\*(1+beta\*\*2)

c[j-1] = omega

d[j] = -(beta\*\*2)\*omega\*(T[i-1,j+1] + T[i+1,j+1]) - (1-omega)\*2\*(1+beta\*\*2)\*T[i,j+1]

T[i][1:col - 1] = tridiagonal\_solve(a, b, c, d)

for i in range(1,row-1):

for j in range(1,col-1):

error += np.abs(T[i,j]-T\_old[i,j])

print('Number of iterations:',iter)

plt.figure(figsize=(4,6))

X,Y = np.meshgrid(np.linspace(0,1,col),np.linspace(0,2,row))

plt.contourf(X,Y,T,100,cmap='jet')

plt.colorbar()

plt.show()

### ADI

import numpy as np

import matplotlib.pyplot as plt

row, col = (41, 21)

T = np.zeros((row,col),dtype=float)

T[40,:] = 100

tolerance = 0.01

dx = 1/(col-1)

dy = 2/(row-1)

beta = dx/dy

def tridiagonal\_solve(a, b, c, d):

n = len(d)

for i in range(1, n):

m = a[i-1] / b[i-1]

b[i] = b[i] - m \* c[i-1]

d[i] = d[i] - m \* d[i-1]

x = np.zeros(n)

x[-1] = d[-1] / b[-1]

for i in range(n-2, -1, -1):

x[i] = (d[i] - c[i] \* x[i+1]) / b[i]

return x

error = 1

iter = 0

a = np.zeros(col-3)

b = np.zeros(col-2)

c = np.zeros(col-3)

d = np.zeros(col-2)

a1 = np.zeros(row-3)

b1 = np.zeros(row-2)

c1 = np.zeros(row-3)

d1 = np.zeros(row-2)

while error > tolerance:

error = 0

iter += 1

T\_old = T.copy()

for i in range(1,row-1):

d[0] = -(beta\*\*2)\*(T[i-1,1] + T[i+1,1]) - T[i,0]

d[-1] = -(beta\*\*2)\*(T[i-1,col-2] + T[i+1,col-2]) - T[i,col-1]

b[0] = -2\*(1+beta\*\*2)

b[-1] = -2\*(1+beta\*\*2)

for j in range(1,col-3):

a[j-1] = 1

b[j] = -2\*(1+beta\*\*2)

c[j-1] = 1

d[j] = -(beta\*\*2)\*(T[i-1,j+1] + T[i+1,j+1])

T[i][1:col - 1] = tridiagonal\_solve(a, b, c, d)

for i in range(1,col-1):

d1[0] = -(T[1,i-1] + T[1,i+1]) - T[0,i]\*(beta\*\*2)

d1[-1] = -(T[row-2,i-1] + T[row-2,i+1]) - T[row-1,i]\*(beta\*\*2)

b1[0] = -2\*(1+beta\*\*2)

b1[-1] = -2\*(1+beta\*\*2)

for j in range(1,row-2):

a1[j-1] = beta\*\*2

b1[j] = -2\*(1+beta\*\*2)

c1[j-1] = beta\*\*2

d1[j] = -(T[j+1,i-1] + T[j+1,i+1])

T[1:row-1,i] = tridiagonal\_solve(a1, b1, c1, d1)

for i in range(1,row-1):

for j in range(1,col-1):

error += np.abs(T[i,j]-T\_old[i,j])

print('Number of iterations:',iter)

plt.figure(figsize=(4,6))

X,Y = np.meshgrid(np.linspace(0,1,col),np.linspace(0,2,row))

plt.contourf(X,Y,T,100,cmap='jet')

plt.colorbar()

plt.show()

### ADI with Relaxation

import numpy as np

import matplotlib.pyplot as plt

row, col = (41, 21)

T = np.zeros((row,col),dtype=float)

T[40,:] = 100

tolerance = 0.01

dx = 1/(col-1)

dy = 2/(row-1)

beta = dx/dy

omega = 1.15

def tridiagonal\_solve(a, b, c, d):

n = len(d)

for i in range(1, n):

m = a[i-1] / b[i-1]

b[i] = b[i] - m \* c[i-1]

d[i] = d[i] - m \* d[i-1]

x = np.zeros(n)

x[-1] = d[-1] / b[-1]

for i in range(n-2, -1, -1):

x[i] = (d[i] - c[i] \* x[i+1]) / b[i]

return x

error = 1

iter = 0

a = np.zeros(col-3)

b = np.zeros(col-2)

c = np.zeros(col-3)

d = np.zeros(col-2)

a1 = np.zeros(row-3)

b1 = np.zeros(row-2)

c1 = np.zeros(row-3)

d1 = np.zeros(row-2)

while error > tolerance:

error = 0

iter += 1

T\_old = T.copy()

for i in range(1,row-1):

d[0] = -(beta\*\*2)\*omega\*(T[i-1,1] + T[i+1,1]) - omega\*T[i,0] -(1-omega)\*2\*(1+beta\*\*2)\*T[i,1]

d[-1] = -(beta\*\*2)\*omega\*(T[i-1,col-2] + T[i+1,col-2]) - omega\*T[i,col-1] -(1-omega)\*2\*(1+beta\*\*2)\*T[i,col-2]

b[0] = -2\*(1+beta\*\*2)

b[-1] = -2\*(1+beta\*\*2)

for j in range(1,col-3):

a[j-1] = omega

b[j] = -2\*(1+beta\*\*2)

c[j-1] = omega

d[j] = -(beta\*\*2)\*omega\*(T[i-1,j+1] + T[i+1,j+1]) - (1-omega)\*2\*(1+beta\*\*2)\*T[i,j+1]

T[i][1:col - 1] = tridiagonal\_solve(a, b, c, d)

for i in range(1,col-1):

d1[0] = -omega\*(T[1,i-1] + T[1,i+1]) - omega\*T[0,i]\*(beta\*\*2) -(1-omega)\*2\*(1+beta\*\*2)\*T[1,i]

d1[-1] = -omega\*(T[row-2,i-1] + T[row-2,i+1]) - omega\*T[row-1,i]\*(beta\*\*2) -(1-omega)\*2\*(1+beta\*\*2)\*T[row-2,i]

b1[0] = -2\*(1+beta\*\*2)

b1[-1] = -2\*(1+beta\*\*2)

for j in range(1,row-2):

a1[j-1] = omega\*(beta\*\*2)

b1[j] = -2\*(1+beta\*\*2)

c1[j-1] = omega\*(beta\*\*2)

d1[j] = -omega\*(T[j+1,i-1] + T[j+1,i+1]) - (1-omega)\*2\*(1+beta\*\*2)\*T[j+1,i]

T[1:row-1,i] = tridiagonal\_solve(a1, b1, c1, d1)

for i in range(1,row-1):

for j in range(1,col-1):

error += np.abs(T[i,j]-T\_old[i,j])

print('Number of iterations:',iter)

plt.figure(figsize=(4,6))

X,Y = np.meshgrid(np.linspace(0,1,col),np.linspace(0,2,row))

plt.contourf(X,Y,T,100,cmap='jet')

plt.colorbar()

plt.show()